BLACK BOX TESTING

Black box testing is a form of functional software testing where the internal structure, design, and implementation of the software are unknown (Figure 1). Since only the input and output values are known, the program is viewed as a black box. The specification of the program's requirements is used as the source of information for determining and designing the tests. The advantage of applying these methods is that tests can be written parallel to the software development because they are developed based on the requirements specification. This way, time is not wasted on testing once the software is already developed. However, there is a possibility that the functionality of the software is not detailed in the specification, as users often do not know all their needs, and requirements change during coding. This complicates the design of the tests.

The disadvantage of this testing method is that not all critical parts of the program are covered, which can lead to various omissions during testing. This method can identify incorrectly implemented functionalities or errors in the program's operation.

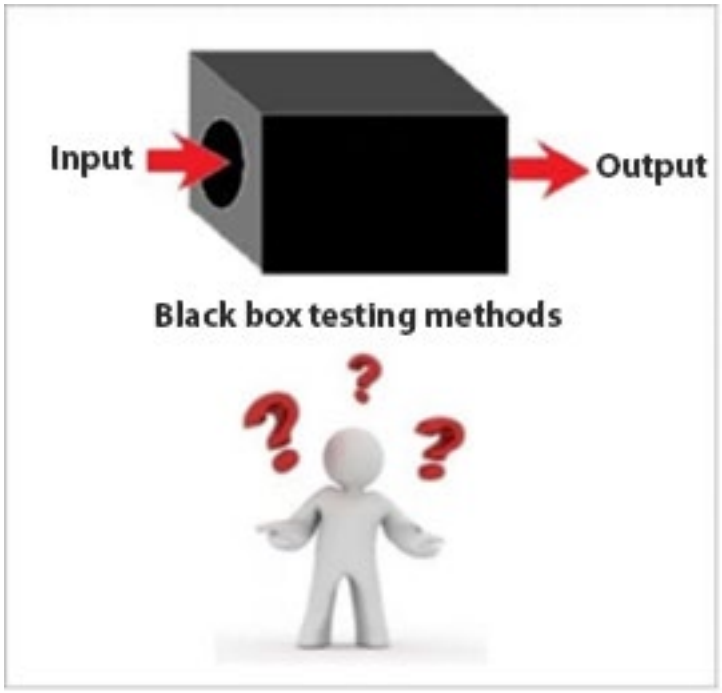

Figure 1. Black Box Testing

The techniques used in black box testing include:

1. Equivalence classes
2. Boundary value analysis
3. Decision tables
4. Cause-effect graph
5. State transition testing

1. Equivalence Class Partitioning

This is one of the basic methods of black box testing and can be applied at any level of testing. The primary goal of this method is to divide input and output values into equivalence classes (i.e., disjoint subsets). The set of input data can be represented by the union of all subsets. It is necessary to test at least one representative value from each class, and if the software behaves correctly, it can be assumed that there are no errors for all other values. All types of variables within the allowed range are analyzed, and legal and illegal equivalence classes are identified for each condition. Legal classes include only valid values, while illegal classes include all other values of the input data.

An example is where the input data is of a specific length, such as a unique identification number that contains 13 digits. The valid value is 13 digits long, while all shorter or longer values are invalid.

Example 1. If the program's input condition is a number between 0 and 100, a legal equivalence class is defined within the interval 0 ≤ number ≤ 100, and two illegal classes are number < 0 and number > 100 (Figure 2).

Solution:

Illegal	**Legal**	**Illegal**
<-1	0 100	>101
Equivalence Class 1	Equivalence Class 2	Equivalence Class 3

Figure 2. Example of equivalence class for an allowed range of 0 to 100

When testing, it is sufficient to take one value from each equivalence class. For example, if testing input values **-10**, **10**, and **110**, the values to be tested from all three classes will be covered.

Example 2. Formulate student grades based on the number of points (integer values) achieved in a subject. If a student has the following number of points:

0 - 50: did not pass

51-60: grade 6

61-70: grade 7

71-80: grade 8

81-90: grade 9

91-100: grade 10.

Solution: In this case, values that are negative numbers and numbers greater than 100 are not analyzed. An experienced tester should also include these values as equivalence classes to ensure coverage of all integer values that the user can enter (Figure 3).

Illegal	Legal	Legal	Legal	Legal	Legal	Legal	Illegal
≤ -1	0 50	51 60	61 70	71 80	81 90	91 100	≥101
EC 1	EC 2	EC 3	EC 4	EC 5	EC 6	EC 7	EC 8

Figure 3. Example of equivalence classes for formulating student grades

It can be observed that the tester must take more values for testing to cover all legal and illegal equivalence classes. In this case, all negative values and values greater than 101 are illegal. For legal values, one value from each equivalence class should be selected and checked to see if the output results are correct (i.e., if the corresponding grade is obtained for each value).

2. Boundary Value Analysis

In practice, it is common for programmers to make mistakes when writing conditional expressions. For example, instead of writing the value $x \geq 0$ when it is necessary to isolate positive values (including 0), $x > 0$ is written, and in such cases, even using the equivalence class method, the error will not be detected. Therefore, it is necessary to check the boundary value during testing (in this case, 0) to ensure that the tests cover all critical values. If the programmer wrote in the code:

if(x ≤ 0) throw new IllegalArgumentException();

0 as a boundary value during testing will throw an exception indicating that an error was made. Therefore, it is necessary to determine the boundaries of different equivalence classes and then take a test for each boundary value.

Tests related to boundary value analysis are associated with equivalence classes because equivalence classes have boundary values that need to be examined.

Example 1. Form equivalence classes and boundary value analysis if a value between 1 and 1000 is required.

Solution: The smallest value is 1, and the largest is 1000. These values are called valid (legal) boundary values. All values between these two values are also valid. Invalid (illegal) values are outside this range. They can only be analyzed, or values around them can be taken. For the given example, in the case of the lower boundary value, values 0, 1, 2 are taken, and for the upper boundary value, 999, 1000, 1001. The table shows the values used for equivalence classes and boundary value analysis.

Equivalence Class

Classification	I	V	I
Partition	<1	1-100	>=101
Value	Error	50	150
Expected Result	R	A	R

Boundary Value Analysis

Classification	I	V	V	V	V	I
Boundary	1			100		
Value	0	1	2	99	100	101
Expected Result	R	A	A	A	A	R

Legend: I – Invalid value V – Valid value R – Rejected value A – Accepted value

Figure 4. Solution representation using equivalence class and boundary value analysis for values between 1 and 1000

Example 2. Form tests to determine if three entered integer values a, b, and c, whose values are in the range of 1-200, represent the sides of a triangle.

Solution:

The integer values a, b, and c must satisfy the following conditions:

$C_{1:}\ 1 \le a \le 200$ $C_{4:}\ a < b + c$

$C_{2:}\ 1 \le b \le 200$ $C_{5:}\ b < a + c$

$C_{3:}\ 1 \le c \le 200$ $C_{6:}\ c < a + b$

The output values for the given problem are equilateral triangle, isosceles triangle, or "The values do not represent the sides of a triangle."

How to generate a test for equivalence classes and boundary value analysis? For the given interval [1, 200], 1 is the lower boundary value, and 200 is the upper boundary value. When testing, the boundary values and the first closest value to the boundaries are taken, as well as some arbitrary value within the given interval, e.g., the middle of the boundary (100). It is known that the program has three input data: a, b, and c. In this case, it is obtained that for n = 3 input data, a minimum of 4*n+1 tests are needed (Table 1).

Table 1. Tests for checking the sides of a triangle

Test	a	b	c	Expected Result
1.	100	100	1	Isosceles
2.	100	100	2	Isosceles
3.	100	100	100	Equilateral
4.	100	100	199	Isosceles
5.	100	100	200	Not a triangle
6.	100	1	100	Isosceles
7.	100	2	100	Isosceles
8.	100	100	100	Equilateral
9.	100	199	100	Isosceles
10.	100	200	100	Not a triangle
11.	1	100	100	Isosceles
12.	2	100	100	Isosceles
13.	100	100	100	Equilateral
14.	199	100	100	Isosceles
15.	200	100	100	Not a triangle

Here we notice that instead of 13 tests, we have 15. Tests numbered 3, 8, and 13 are the same. We will ignore them and keep only test 3, so we get 13 tests.

3. Decision Table

A decision table is a significant technique when different combinations of input parameters are required during testing. It is suitable for use with complex business logic as it helps achieve better test coverage. This technique is simple to apply and interpret. It can be used for developing functionalities to better understand different combinations. As the number of input data increases, the system becomes more complex for analysis. Thus, the number of combinations for n input parameters becomes 2^n output values.

The advantages of this method are in cases where the system behaves differently with the same input values. This table creates efficient combinations and better test coverage. If the number of input data is small, the test coverage is 100%.

The main drawback of this technique is that with an increase in the number of input parameters, the table becomes more complex and difficult to track. Decision tables are good for:

- Covering requirements containing logical conditions
- Documenting system design
- Recording complex business rules applied to the system
- As a guide for creating tests that otherwise cannot be used.

Each row in the decision table represents a state or action. In the table columns, business rules (Rules-R) are listed, defining unique combinations of states. These states influence the execution of actions related to a specific rule. When designing tests, it is necessary to have at least one test per column, usually covering all combinations of activated states. This method can be applied in all situations where software execution depends on several logical decisions.

Example 1. Create a scenario that creates a decision table for solving the problem of login using a user's email address and password. If the user correctly enters the Email and Password, they will be logged into a specific application. If any parameter is entered incorrectly, an error is displayed.

Rule 1 - Incorrect Email and Password

Rule 2 - Correct Email and Incorrect Password

Rule 3 - Incorrect Email and Correct Password

Rule 4 - Correct Email and Password

Solution:

Table 2. Decision table for Email and Password input

Condition	Test 1	Test 2	Test 3	Test 4
	Rule 1	Rule 2	Rule 3	Rule 4
Email (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Izlaz (E/H)	E	E	E	H

Legend:

T – Correct Email/Password

F – Incorrect Email/Password

E – Error message

H – Display home screen

When the table is converted to tests, two scenarios can be created:

1. Correct Email and Password entry and the expected result is logging into the application's home screen.
2. One of the following scenarios:
    - Incorrect Email and Password entry and clicking on login, expecting two error messages.
    - Incorrect Email and correct Password entry and clicking on login, expecting an error message for incorrect Email.
    - Correct Email and incorrect Password entry and click on login, expecting an error message for incorrect Password.

Example 2. Create a scenario to determine if a child attends primary school, high school, or university based on age.

Rule 1 - Students aged 6-14 attend primary school

Rule 2 - Students aged 15-19 attend high school

Rule 3 - Students older than 19 attend university

Rule 4 - Gifted students aged over 16 can attend university.

Solution:

Table 3. Decision table for determining school attendance based on age

Condition	Test 1	Test 2	Test 3	Test 4
	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**
Age 6-14	T	F	F	F
Age 15-19	F	T	F	?
Age over 19	F	F	T	?
Gifted with 16 and more years	?	F	?	T
States/Actions				
In primary school	T	F	F	F
In high school	F	T	F	F
In university	F	F	T	T

Legend:

T-True for business rule **F**-False for business rule **?**-Determined by student

4. Cause and Effect Graph

A cause-effect graph (Ishikawa diagram) is used for a visual representation of the relationship between input data (causes) and output data (effects) using logical relations expressed by Boolean algebra. When generating tests from the graph, different combinations of input values are selected for testing. This technique is often used in combination with decision tables. Steps in forming a cause-effect graph:

1. Identify and describe input data (causes) and output data (effects).
2. Create the cause-effect graph.
3. Convert the graph into a decision table.
4. Form tests based on the decision table. The columns of the decision table represent tests.

Logical functions used when drawing a cause-effect graph:

- **Identity** – if node a has a value of 1, then node b has a value of 1; otherwise, both nodes have a value of 0
- **Not** – if node a has a value of 1, node b has 0, and vice versa
- **Or** – if at least one of the input nodes has a value of 1, the output node also has a value of 1; otherwise, the output node is 0
- **And** – if all input nodes have a value of 1, the output node has a value of 1; otherwise, the output node is 0

Four basic symbols for representing dependencies between inputs and outputs are given in Figure 5.

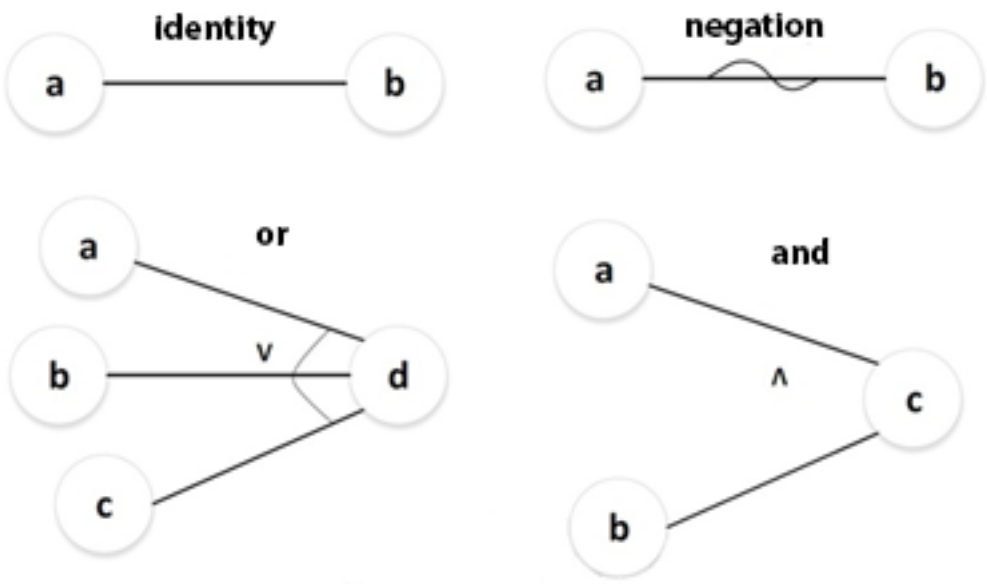

Figure 5. Symbols of cause and effect dependencies

Example 1. Create a cause-effect graph, decision table, and write tests to determine if given values a, b, and c form the sides of a triangle and if so, whether they represent a regular, isosceles, or equilateral triangle.

Solution:

In the first step, input states (causes) and results or actions (effects) are identified and described.

Causes are marked with the letter C (Cause):

C1: Side a is less than the sum of b and c

C2: Side b is less than the sum of a and c

C3: Side c is less than the sum of a and b

C4: Side a is equal to side b

C5: Side a is equal to side c

C6: Side b is equal to side c

Effects are marked with the letter E (Effect):

E1: Not a triangle

E2: Regular triangle

E3: Isosceles triangle

E4: Equilateral triangle

E5: Impossible

In the second step, the cause-effect graph is created (Figure 6). Converting the cause-effect graph into a decision table is shown in Table 4.

Figure 6. Cause-effect graph for testing triangle sides

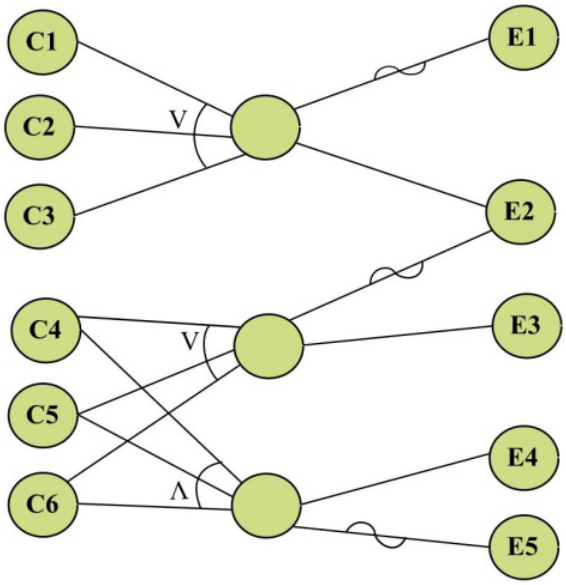

Table 4. Decision table for testing triangle sides

Uslovi	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
C1: a < b+c?	0	1	1	1	1	1	1	1	1	1	1
C2: b < a+c?	X	0	1	1	1	1	1	1	1	1	1
C3: c < a+b?	X	X	0	1	1	1	1	1	1	1	1
C4: a=b?	X	X	X	1	1	1	1	0	0	0	0
C5: a=c?	X	X	X	1	1	0	0	1	1	0	0
C6: b=c?	X	X	X	1	0	1	0	1	0	1	0
E1: Not a triangle	**1**	**1**	**1**								
E2: Regular triangle											**1**
E3: Isosceles triangle							**1**		**1**	**1**	
E4: Equilateral				**1**							
E5: Impossible					**1**	**1**		**1**			

Table 5. Test table for testing triangle sides

Test	a	b	c	Expected Result
1	10	5	2	Not a triangle
2	5	10	2	Not a triangle
3	5	2	10	Not a triangle
4	5	5	5	Equilateral
5	-4	3	6	Impossible
6	6	-2	4	Impossible
7	2	2	3	Isosceles
8	5	3	-5	Impossible
9	2	3	2	Isosceles
10	3	2	2	Isosceles
11	3	4	5	Regular

The result is 11 tests based on 11 rules (Table 5).

5. State Transition Diagram

This technique is based on the specification of software behavior in the form of a finite automaton. It is applied only to software whose behavior can be represented by a finite automaton. This means that the observed system can be in a finite number of different states, and transitions from one state to another are determined by the automaton's rules. Any system where different outputs for the same input data are obtained depending on previous events can be represented by a state model graph.

The system can remain in a state for an arbitrary length of time until an external event occurs, causing a state change. The transition between states is determined by the current state of the system and the input event. An event is an external occurrence that triggers a transition to a new state and can be triggered in various ways.

Example 1. Draw a state diagram for the login page of an application that locks the account after three incorrect entries of the username and password.

Solution:

The state diagram for the defined scenario is shown in Figure 8.

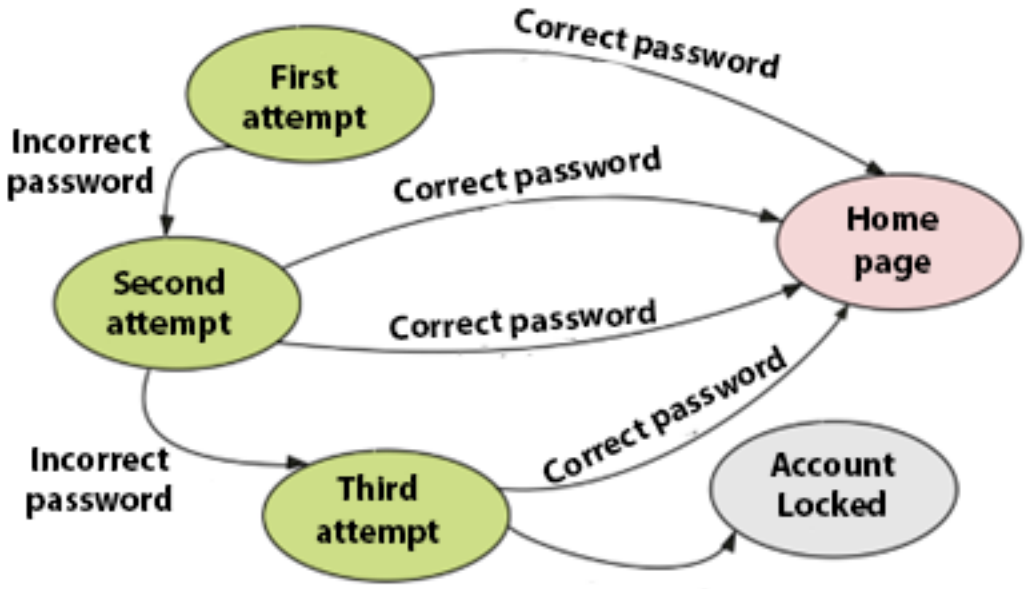

Figure 8. State diagram for the login page of the application

Based on the state diagram, a state table is created (Table 10).

Table 10. State table for the login page of the application

State	Login	Correct Password	Incorrect Password
S1	First attempt	S4	S2
S2	Second attempt	S4	S3
S3	Third attempt	S4	S5
S4	Home page		
S5	Display message "Account is locked. Contact the administrator."		

Entering the correct password on the first or second attempt redirects the user to the home page (state S4).

Entering an incorrect password on the first attempt displays a message to try again and redirects the user to state S2 for the second attempt. Entering an incorrect password on the second attempt displays a message and redirects the user to state S3 for the third attempt.

Entering an incorrect password on the third attempt redirects the user to state S5 and displays the message "Account is locked. Contact the administrator."